Date: August 17, 2000 Express Mail Label No. EL551544495US

Inventors: Lory Molesky and Robert Hanckel

Attorney's Docket No.: 1958.2005-000

INTERVAL-BASED ADJUSTMENT SYSTEM FOR DATABASES

BACKGROUND

For many applications, data is stored in a database as a time series of data values. The data retrieved from the database may need to be adjusted for events. In particular, it may be desirable to apply an interval-based adjustment (IBA) to the data.

Examples of interval-based adjustments include the application of stock splits and currency conversions. Such adjustments may also be used to reflect dividend distributions from stocks and mutual funds. These adjustment triggers are typically infrequent events.

To reduce rounding and precision errors, the time series data is typically stored as unadjusted data values. When the data is stored in its unadjusted form, the data would be adjusted on retrieval to reflect the adjustments. For example, on retrieval of stock prices, one may like to see all data adjusted based on the current price taking into consideration the stock's split and dividend history. Using the adjusted data, the performance of the security over time can be accurately ascertained and presented to the user. Typically, the adjustments are applied using custom programming.

SUMMARY

The custom programming approach has at least two disadvantages. In the first place, each installation may have to write custom programs to solve the problem. This need to develop custom code for each database increases development costs. In addition, the performance of the resulting custom program tends to be slow. A

contributor to the slow performance is the use of multiple scans of the data to first retrieve the data and then to apply any adjustments. Another contributor is the additional memory overhead to store the date while it is being processed.

A better approach exploits the high performance of a database engine. The database engine can retrieve and apply the appropriate adjustment factors for each data element in a time series. In particular, the database engine is a relational database engine. From a user's perspective, the adjustment can be applied transparently.

In a particular embodiment, a database stores a plurality of raw data values organized as a series in a first database structure. The data can be a time series of data, such as financial data. A plurality of intervals of adjustment data are stored in a second database structure. The adjustment data includes an adjustment value for each interval. Furthermore, the adjustment values can be dynamically updated to reflect additional intervals of adjustment data being added to the database.

For a user to query the database for adjusted data values, the first and second database structures are associated. This can be done by creating a view of the database that includes the two structures. This can be done by a system administrator. The user query then applies the adjustment values to the raw data values to yield adjusted data for presentation to a user.

The database can also store data for pending adjustments. In particular, the second database structure can include adjustment data for the pending adjustment. In response to a query for adjusted data using the pending adjustment, an adjustment value for each interval can be computed using the pending adjustment data and applied to the raw data values.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the system will be apparent from the following more particular description of embodiments, as illustrated in the accompanying drawings in which like reference characters refer to the same parts

throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

FIG. 1 is a graphical representation of a stock data time series.

FIG. 2 is a block diagram of a computing system for applying interval-based adjustments.

FIG. 3 is a block diagram of the database tables of FIG. 2.

FIG. 4 is a block diagram illustrating the tables of FIG. 3 populated with data for the chart of FIG.1.

FIGs. 5A-5C are block diagrams illustrating the recording of stock split data in the splits table of FIG. 4.

DETAILED DESCRIPTION

Embodiment of the system can be practiced on various time series data models, including, but not limited to, stock splits, dividend distributors, currency conversion, and expense adjustments. For ease and clarity of description, the system will be described with reference to stock splits.

FIG. 1 is a graphical representation of a stock data time series. The graph illustrates hypothetical time series data, such as daily closing prices of a stock having the ticker symbol “ACME”. As illustrated, the stock has undergone two stock splits, a 3-for-1 split effective on a particular date, D2, and a 2-for-1 split effective on a later date, D4. A user may want to adjust the stock prices to account for the splits. This can be done by either looking forward from a date or looking backward from a date.

As an example of a forward-looking adjustment, a user may want to determine the current value of a share of stock purchased on a particular date, D1, D3, D5. A share purchased on the first date D1 at $30/share would have a value of $30 on that date, D1. At a subsequent date D3, the stock closed at $80/share. Because of the 3-for-1 split on the first split date D2, the user would have 3 shares, instead of 1, valued at $240. The investment has increased 8-fold. At a still later date D5, the stock closed at $50/share. Because of the 3-for-1 split on the first split date D2 and the 2-for-1 split on

the second split date D4, the user would have 6 shares, valued at $300. The investment has now increased 10-fold. From the adjusted data, the user can determine the gain of the investment over time.

One example of adjusting backward in time is a moving average. The discontinuity created by the splits should be adjusted to obtain an accurate moving average. To obtain adjusted data, the prices between the split dates D2, D4 is divided by the second split factor, 2. For example, the stock at date D3 is valued at $40/shore in the current valuation. Price data before the first split date D2 is divided by the first and second split factors, 3*2=6. For example, the stock at date D1 is valued at $5/share in the current valuation.

FIG. 2 is a block diagram of a computing system for applying interval-based adjustments. As illustrated, a server computer 10 is interconnected to one or more client computers 20a,...,20N via a communications medium 5. The communications medium 5 can be any suitable medium, including wired or wireless. The communications medium 5 can directly connect the client 20 and server 10, or it can be a public communications system, such as the Internet. It should be understood, that a client-server architecture is not required for the system, the methods described herein can also be employed in a standalone computer system and in multi-tiered computing systems.

The server 10 can be a single computer or a plurality of clustered computers. The server 10 includes an operating system (OS) 12 and a database engine 14 layered over the operating system 12. Together the operating system 12 and the database engine 14 cooperate to manage a database 16. In particular, the database 16 is a relational database that stores time-series data and the database engine 14 is Oracle8*i*, commercially available from Oracle Corporation, Redwood Shores, California.

The database 16 can include three relational database table structures. A map table 18 maps a reference symbol to an adjustment table 19. In the particular example described below, the time-series data is daily raw (i.e., unadjusted) stock price data and the adjustments are stock splits. For this example, the reference symbol is a stock ticker symbol. The map table 18 thus maps the ticker symbol to splits table 19.

Database queries can be initiated by a client computer 20a. The server 10 uses the database engine 14 to retrieve data from the database 16 to satisfy the query. Results are returned to the client computer 20a for display to a user. These results are automatically adjusted using data from the adjustment table 19.

Methods for applying the adjustment data exploit the high performance of the relational database engine 14 to retrieve and apply the appropriate split factor for each element in a pricing time series. Furthermore, from the user's perspective, applying the split factor is performed transparently.

FIG. 3 is a block diagram of the database tables of FIG. 2. The stock table 17, as shown, includes five fields. The stock's ticker symbol is stored as text in a symbol field 17-1. The date of the trade recorded is stored in a timestamp date field 17-2. The high, low, and closing prices of the stock are stored in three numeric fields: high 17-3, low 17-4, and close 17-5. The stock table 18 includes one record for each day the stock is traded, keyed by the symbol field 17-1.

The map table 18, as shown, includes two fields: a symbol character field 18-1, and an adjustment table field 18-2. The symbol field 18-1 stores the stock ticker symbol. The adjustment table field 18-2 can be a text field or a pointer field that dereferences to the splits table 19 for the stock identified in the symbol field 18-1..

The splits table 19, as shown, includes five fields. Each record in the splits table 19 is associated with one interval. An effective_date date field 19-1 records the effective date of an adjustment (split) and an end_date date field 19-2 records the date of the next split adjustment. The split information for the interval between the effective date and the end date is recorded in the numerator integer field 19-3 and the denominator integer field 19-4. The factor value to convert prices during this interval period to current valuations is stored in the factor numeric field 19-5.

Every time a new entry is entered into the splits table 19, reflecting a new split, the factor field 19-5 for each entry is recalculated. The factor is computed from the cumulative products of the numerators and denominators as follows:

$$\text{factor}_i = \frac{\prod_{j=i+1}^{N} \text{numerator}_j}{\prod_{j=i+1}^{N} \text{demonimator}_j}$$

where i is the entry position in the splits table 19, beginning from 1; and

N is the total number of entries in the splits table 19.

In a particular embodiment, the recalculation of the factor fields 19-5 are performed by the database engine 14 (FIG. 2) in response to the addition of a new table entry.

FIG. 4 is a block diagram illustrating the tables of FIG. 3 populated with data for the chart of FIG.1. For convenience, only the closing price data for the security "ACME" is shown in the stock table 17. The map table 18, associates the security "ACME" with an adjustment table "ACME_SPLITS_TABLE". In the splits table 19, the initial effective date, for the first interval (i = 1), is set equal to a sufficiently low data such as the lowest limit of the data fields, MINDATE. Likewise, the end date for the last interval is set equal to MAXDATE, which can be the upper limit of the date field. Newly announced splits are recorded in the splits table 19. As shown in the splits table 19, pricing data during the first interval must be factored by 6 to be expressed in current valuations.

FIGs. 5A-5C are block diagrams illustrating the recording of stock split data in the splits table of FIG. 4. The initial condition, before any split, is illustrated by FIG. 5A. As shown, there is one entry (N=1), which directs the system to retrieve data as it is, essentially by multiplying all prices between MINDATE and MAXDATE by 1.

After the first (3-for-1) split, the splits table 19 is illustrated by FIG. 5B. As shown, the splits table 19 has two entries (N=2), which direct the system to divide prices by 3 for dates between MINDATE and D2 and to divide prices by 1 for dates between D2 and MAXDATE.

After the second (2-for-1) split, the splits table 19 takes on its final form, as also illustrated by FIG. 5C. As shown, the splits table 19 has three entries (N=3). The

system divides prices by 6 for dates between MINDATE and D2, divides prices by 2 for dates between D2 and D4, and divides prices by 1 for dates between D4 and MAXDATE.

Structured Query Language (SQL) queries reference the splits table 19 and the stock table 17 to automatically perform price adjustments. This price adjustment is implemented with a view, which allows the transparent application of the adjustments. The SQL query makes use of the effective date field 19-1 and the end date field 19-2 to automatically and efficiently apply the appropriate split factor upon retrieval of the price date. In particular, the effective and end date columns 19-1, 19-2 are used in retrieval queries' WHERE clause to select the appropriate price adjustment from the splits table 19.

In a particular embodiment using Oracle8*i*, the splits table 19 is implemented as a nested table (SplitSeriesTab) of records (SplitSeriesCells). The queries posed by the user to retrieve data adjusted for splits are expressed almost identical to a SQL query that would retrieve unadjusted data. Specifically, to retrieve data adjusted for splits, the user substitutes the name of the underlying table holding the pricing data with a view. That is, the

```
SELECT * FROM stock_table
```

clause to retrieve unadjusted data, is rewritten as:

```
SELECT * FROM adjusted_view
```

to retrieve the prices adjusted for splits.

The view that accomplishes the adjustment for the price table having high and close pricing columns is:

```
CREATE VIEW adjusted_view (symbol, timestamp, close, high);
        AS SELECT   stock.symbol,
                    stock.timestamp,
                    stock.close / splits.factor,
                    stock.high / splits.factor
            FROM stock_map map, stock_table stocks,
```

```
    TABLE (cast
        (map.sf.series as TSDEV.SplitSeriesTab)) splits
WHERE map.symbol = stocks.symbol AND
    stocks.timestamp > splits.effective_date AND
    stocks.timestamp < splits.end_date;
```

Pricing data selected from the stock table 17 is adjusted by:

1) dividing the price by the factor column of the splits table 19;
2) the appropriate split factor is determined by the WHERE clause, which chooses the split factor based on the timestamp of each pricing row; and
3) in the implementation of this view, stock_map is used to support the selection of various securities.

This view-based pricing adjustment is extremely efficient because it exploits simple range query technology of the database engine that references indexes based on the date column of the pricing table.

An administrator may also want to insert "pending splits" into the splits table. Pending splits are those that have been announced but have not yet become effective.

The previous solution relies on a split factor column that is always current, that is, the split factor column must be recomputed prior to the first retrieval of unadjusted data on or after the effective date of a new split.

There are many ways to enforce this property, such as:

-Only insert new splits on the effective date of the split. On insert, recompute the split factor column based on the cumulative product of the numerator and denominators.

-Allow pending splits to be inserted into the splits table, but define a time-based trigger to recompute the split factor column at the effective_date of the pending split.

Both of these solutions require that the administrator set up the proper maintenance scripts.

Below is an alternative solution that handles pending splits, but does not require additional maintenance scripts. The solution is based on a view that recomputes the proper split factors based on the current date:

```
CREATE VIEW estimate_adjusted (symbol,
                                   timestamp, close, high);
AS SELECT   stocks.symbol,
            stocks.timestamp,
            stocks.close / (n/d),
            stocks.high / (n/d)
  FROM stock_map map, stock_table stocks,
       (SELECT PROD (split2numerator) n, PROD (split2denomerator) d
       FROM TABLE
                stock_map map2, stock_table stocks2,
                (cast (map.sf.series as TSDEV.SplitSeriesTab)) splits2
            WHERE map2.symbol = stocks2.symbol AND
            splits2.effective_date < SYSDATE;)
            stocks2.timestamp > splits2.effective_date AND
            stocks2.timestamp < splits2.end_date;
     WHERE map.symbol = stocks.symbol
```

Here, SYSDATE is the current date/time of the computing system. This view is similar to the previous view, but extended with a subqueury. The subqueury computes the cumulative product of numerators and denominators from the splits table.

It may also be desirable to apply multiple interval-based adjustments to time series data. For example, a stock price may be adjusted due to both splits and distributions. The database engine 14 can handle these multiple adjustments, either in parallel or series. First, the map table 18 can have more than one column per stock, one

dereferencing to the splits table and another dereferencing to a distribution table. To apply the adjustments in parallel, a view can be created using the multiple adjustment tables. Applying the adjustments in series may be more difficult because the adjustments should be applied in time order.

Those of ordinary skill in the art will recognize that methods involved in the interval-based adjustment system may be embodied in a computer program product that includes a computer usable medium. For example, such a computer usable medium can include a readable memory device, such as a solid state memory device, a hard drive device, a CD-ROM, a DVD-ROM, or a computer diskette, having computer readable program code segments stored thereon. The computer readable medium can also include a communications or transmission medium, such as a bus or a communications link, either optical, wired, or wireless, having program code segments carried thereon as digital or analog data signals.

While this system has been particularly shown and described with references to particular embodiments, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the scope of the invention encompassed by the appended claims.